White Paper Report

Report ID: 106511

Application Number: HD5155912

Project Director: George Williams (GWilliams@uscupstate.edu)

Institution: University of South Carolina Research Foundation

Reporting Period: 9/1/2012-8/31/2013

Report Due: 11/30/2013

Date Submitted: 3/12/2014

Making the Digital Humanities More Open

September 2013

Project Directors: George H. Williams and Jennifer Guiliano

Project #: HD-51559-12
Term Length: 1 September 2012 - 31 August 2013

White Paper Authors: James Smith, Cory Bohen

Summary: The BrailleSC project received Level 2 startup funding ($49,339) to undertake its second stage of development by designing and deploying a WordPress-based accessibility tool designed to create braille content for endusers who are blind or low vision.

1. Project Overview

Funded by a Level 2 Digital Humanities Start-Up Grant, the BrailleSC.org project undertook its second stage of development by designing and deploying a WordPress-based accessibility tool enabling end users with a variety of disabilities and abilities to experience and contribute to online resources. Specifically, the project extended the use of Anthologize, a free and open source plugin for WordPress that translates WordPress text into PDF, ePub, and TEI, to include conversion of text to Braille, both SimBraille and a format suitable for embossing. As part of the same work, the project provided the tools for making Braille available visually through the WordPress interface.

This white paper provides an overview of the project, identifies areas for improvement and lessons learned, and outlines possible next steps. An appendix includes much of the documentation on the Braille formats and using LibLouis from PHP as compiled during the course of the project.

All software constructed during this project is available for free and as open-source code. The WordPress plugin is also published in the WordPress plugin index and available for installation from within the WordPress administrative interface.

Our project is a collaboration among scholars from two different institutions—the University of South Carolina Upstate and the University of Maryland, College Park—and from a variety of disciplines:

- **George Williams**, the project director, is an associate professor of English at the University of South Carolina Upstate who acted as project coordinator and facilitator.
- **Tina Herzberg**, associate professor of education and director of the Special Education–Visual Impairment Program at the University of South Carolina Upstate, provided feedback on the correctness of the Braille translations of sample English text.
- **Jennifer Guiliano**, assistant director of the Maryland Institute for Technology in the Humanities (MITH) at the University of Maryland, College Park, acted as liason between the Upstate project members and MITH, which provided the technical development.
- **James Smith**, software architect of the Maryland Institute for Technology in the Humanities (MITH) at the University of Maryland, College Park, provided oversight of the technical development process.
- **Cory Bohon**, an independent developer, coded the LibLouis remote web service and

the PHP libraries for accessing the LibLouis software.

- **Amanda Visconti**, graduate student in English at the University of Maryland and webmaster for the Maryland Institute for Technology in the Humanities (MITH) (at the time of the project work), developed the WordPress plugin framework.

2. Project Activities

The primary project activities consisted of building a WordPress plugin that provided a Braille translation option for Anthologize and a Braille translation service that would allow the WordPress plugin to provide the translation without requiring the administrator of the WordPress site to also install the LibLouis Braille translation library. The project successfully completed both activities with limited success: the WordPress plugin is available and works as planned, but the translation service may not be written suitably for deployment as a public service available to everyone who might use the WordPress plugin.

Due to the loss of one of the developers at MITH, we hired on a part-time basis an outside contract developer to write the PHP libraries and Braille to English translation service. We had our in-house, part time website developer write the WordPress plugin.

We employed an agile development methodology which focused on ensuring the completion of the highest priority objectives: allowing the Braille translation of content produced through the Anthologize WordPress plugin. A secondary priority was providing a publicly accessible translation service. We also used a layered approach in which we built several relatively self-contained components that together allowed us to achieve our primary objective.

Instead of writing an Anthologize plugin directly, we chose to create a WordPress plugin that would allow the translation of selected parts of a WordPress website. This allowed us to test the translation services without going through the Anthologize plugin. Building a WordPress plugin first also made for an easier development target for our junior developers.

We designed the WordPress plugin to work with the LibLouis Braille to English translation software. This software is not written in PHP, so we had to write a PHP library that could interface with the command line tool provided by the LibLouis package. Because we knew that we wanted to allow the WordPress plugin to work without requiring the installation of LibLouis on the same server as the WordPress site, we developed a second PHP library that had the same interface as the first, but would use a remote web service to provide the translation. The WordPress plugin was designed to work with either library with the selection made through the settings page.

The remote service was written using the Sinatra framework (http://www.sinatrarb.com/) designed for easily and quickly building web services in Ruby. Sinatra-based web services are more scalable than other alternatives, such as CGI scripts, but require more attentive tuning when used as the basis for publicly available web services. See the Evaluation section below for more information on how we intend to address the scalability of this service.

After building the WordPress plugin and ensuring that it could translate English text into Braille using the remote web service, we added the capability for the WordPress plugin to detect the Anthologize plugin and provide a Braille output format for Anthologize. This output format uses the same local or remote translation service as configured in the Braille WordPress plugin settings.

3. Accomplishments

The project aimed to impact several areas of web development and scholarly projects: increase participation by all people in experiencing and creating scholarly digital projects, provide correctly formatted Braille to provide clarity to the reader and allow the Braille reader to easily navigate the materials, and provide free online Braille translation services to the public. All of these goals were met to some degree.

We were able to provide a WordPress plugin that interfaces with Anthologize to provide Braille output as SimBraille for display or in the embossing format used to produced embossed versions of the text. The WordPress plugin and Anthologize integration provide opportunities for increased participation by providing a Braille translation of WordPress content in several forms, though the configuration and provisioning of the WordPress plugin requires access to non-Braille forms of the WordPress administrative interface.

The Braille translation provided by the WordPress plugin and the English to Braille translation service is dependent on the correctness of the translation provided by the LibLouis software. We have noted a few problems in translating highly formatted or typographically complex text in the Evaluation section below.

We produced a English to Braille translation service as open source software that can be installed by anyone with access to a Unix system, though the LibLouis software used to provide the translations is easier to install on some systems than others based on in-house experience. We did all of our testing and development in Apple OS X and Ubuntu Linux environments without any problems installing all of the required software. As a result of the in-house evaluation of this service, we plan to rewrite parts of the translation service so that we can offer a publicly available translation service that doesn’t require significant server tuning.

4. Audiences

The project addressed several different audiences with its products. The primary audience, of course, is the reader of the Braille, but secondary audiences include the WordPress site administrator and content providers.

The primary audience of the project is the Braille reader. The resulting Braille is twofold: the SimBraille equivalent using Unicode code points is designed for a sighted audience learning Braille or reviewing the Braille translation on a display surface; the embossable Braille output is designed for embossing on paper or other interface designed for tactile sensing that can be read by anyone familiar with embossed Braille.

The secondary audience of the project, and the primary audience of the WordPress plugin as an interactive object is the WordPress site administrator. The settings and the Anthologize output format are designed to be used by someone who is already able to interact with the rest of the administrative parts of a WordPress site with Anthologize. The plugin is distributed in the same manner as any other WordPress plugin, allowing any site administrator to install the plugin through the WordPress administrative interface. Configuration of the plugin allows the site to use a locally installed copy of the LibLouis software or, in hosting configurations that do not allow local installation of additional software, configuration of a remote service that provides access to the LibLouis software.

5. Evaluation

We have not sought outside evaluation of the work products. However, in-house evaluation by developers not involved in the work has led to a few opportunities for improvement.

WordPress Plugin

The WordPress plugin provides access to all of the functionality required to provide an English to Braille translation of Anthologize projects and material within a WordPress site. However, the plugin does not allow selection of the language of the text (it assumes that the text is English text) or the ability to restrict access to a remote translation service by means of an API key or similar restriction. Both of these are being considered for future versions of the plugin and service.

Since publishing the WordPress plugin, it has been downloaded 97 times as of Thursday, 5 September 2013.

English to Braille Translation Service

The original grant stipulated that the English to Braille translation web service would be written in PHP. However, we were uncomfortable with this and chose to write it in Ruby instead using the Sinatra framework (http://www.sinatrarb.com/). This translation web service is sufficient to support small web sites such as a personal blog or a Moodle installation supporting a small number of courses. As written, the web service can support a few tens of thousands of requests a day. The MITH staff involved in the evaluation did not consider this sufficient for building a general-access, public service, so we are not offering such a service.[1] As a result of this evaluation, we plan to create a version of the web service that is acceptable to MITH staff so that we can offer the service to the public.

Service Scalability

The primary problem in building the service using Sinatra and LibLouis is the lack of bindings allowing the use of the LibLouis software from within the Ruby environment. Instead, we must execute a separate program for each requested translation, resulting in a substantial amount of overhead that limits the scalability of the service. Rewriting the service in Scala, a language that compiles to the Java virtual machine (JVM), allows the service to use the LibLouis software directly by including the software in the same program that processes the translation requests from the WordPress plugin.

Translation Effectiveness

The translation service uses LibLouis to provide the actual translation using the proper mode for English text. This mode works well with simple ASCII text, but fails when encountering typographical elements expressed in Unicode, such as m-dashes. There is no easy way to discern mathematical text from non-mathematical text, so certain conventions in mathematical Braille might not be observed. Rendering lists in Braille may also pose some formatting issues.

Overall, the translation service provides a good quality rendering in Braille of narrative English text. Advanced typography and non-narrative formatting tend to present problems for the translation process provided by the LibLouis software.

6. Continuation of the Project

In addition to future grant-funded projects, the software produced during the work has

[1] It is worth noting that the result of this evaluation is not dependent on the implementation language of the service: whether the service was written in PHP as stipulated in the grant or in Ruby as actually written, the service would be using the LibLouis software in a way that would result in the perceived weakness surfaced by the evaluation.

been used to provide a Moodle filter that allows questions for students and other content to include SimBraille.

7. Long Term Impact

The WordPress plugin and other software produced as part of the work laid the foundation for further accessibility work at MITH. For example, we are developing a jQuery plugin that can support 6-key input, allowing web forms to have text fields that allow SimBraille input. This software becoming more widely known and used will allow Braille to become a more commonly supported input and display form.

Additionally, the availability of a WordPress plugin removes a reason for WordPress sites not to provide a Braille translation of content, especially in an educational context. Sites that use Anthologize are able to produce Braille versions of their content with little additional effort.

8. Grant Products

The work resulted in the following software:

- A WordPress plugin providing Braille translation of English text. This plugin also provides a SimBraille and embossable file translation of English text for Anthologize, a WordPress plugin that allows the aggregation of WordPress content in the form of a book. The Braille plugin is available at http://wordpress.org/plugins/braille/.
- A set of PHP libraries for interacting with the English to Braille translation service as well as a locally installed copy of the LibLouis software. These PHP libraries have been used in a Moodle filter and can be used in other web site systems such as Drupal. These libraries are available at https://github.com/umd-mith/braille/tree/master.
- An example English to Braille translation web service. The source code for this service is available at https://github.com/umd-mith/braille/tree/master/remote-liblouis. This service is relatively easy to set up and run on an Ubuntu Linux server such as one running through the Amazon EC2 cloud server environment.

9. Lessons Learned

In the course of the work and its evaluation, we learned the following takeaways that will inform any future work that we do. Additional documentation has been added as Appendix B that shares the research into how Braille is represented in a computer and how to use the LibLouis translation software.

- Sinatra allows for rapid development, but does not allow simple scaling for wide public use of the resulting service, especially when we must call a separate program to perform the work represented by the web request to the sinatra framework.
- The text provided to the LibLouis software needs to be cleaner than is obvious from the claim that the software can process HTML. Automated translation has limits that become apparent when translating complex text. As a result, we need to manage expectations while we find ways to improve the translation process and results.

Appendix A. Development Calendar

Github Setup (1 - 12 Oct) [Jim]

Setup github repository supporting BrailleSC work, especially with Wordpress Plugin and related sub-projects.

Library Requirements (1 - 19 Oct) [Cory]

Research LibLouie and PHP to see how the two might be able to work together to produce English-to-Braille translations in a web environment, preferably using UTF-8 instead of alternate typefaces.

Local LibLouie (22 Oct - 21 Dec) [Cory]

Create a PHP library that allows WordPress to use LibLouie without calling out to a remote service. The library should have a sufficiently flexible API to provide access to the affordances surfaced in the previous research milestone. This library will be used in subsequent milestones by Amanda as she develops the Wordpress plugin proper.

The PHP library should be a standalone package that can be installed on a system and used outside of WordPress.

Wordpress Plugin Requirements (29 Oct - 2 Nov) [Amanda]

Research how Anthologize works, how WordPress plugins are designed, and how a WordPress plugin can modify the Anthologize output formats. Install a practice WordPress plugin as a test to create the skeleton for the plugin to built later.

Wordpress Plugin Presentation (5 Nov - 7 Dec) [Amanda]

Produce a basic text filter for WordPress. This will form the core of the plugin and testing as we incorporate the Braille output format into Anthologize.

Remote Library Requirements (14 Jan - 1 Feb) [Cory]

Create an Ubuntu server for development/testing and create a basic Sinatra application. Research calling out to the shell securely from Ruby.

Wordpress Plugin Configuration (11 Feb - 8 Mar) [Jim]

Create the various administrative settings needed to configure the WordPress plugin.

Remote LibLouie (4 Feb - 8 Mar) [Cory]

Create the remote service software that provides access to the LibLouie software as a web service.

Local/Remote LibLouie and Wordpress (11 Mar - 19 Apr) [Cory & Jim]

Test the WordPress plugin and ensure that it can be configured to work with the LibLouis software installed locally as well as the remote service.

Plugin Publication (22 Apr - 31 May) [Jim & Cory]

Publish the WordPress plugin to the WordPress plugin repository and index.

Appendix B: Research Documentation

The material in this appendix is from documentation compiled by Cory Bohon as part of the Library Requirements milestone.

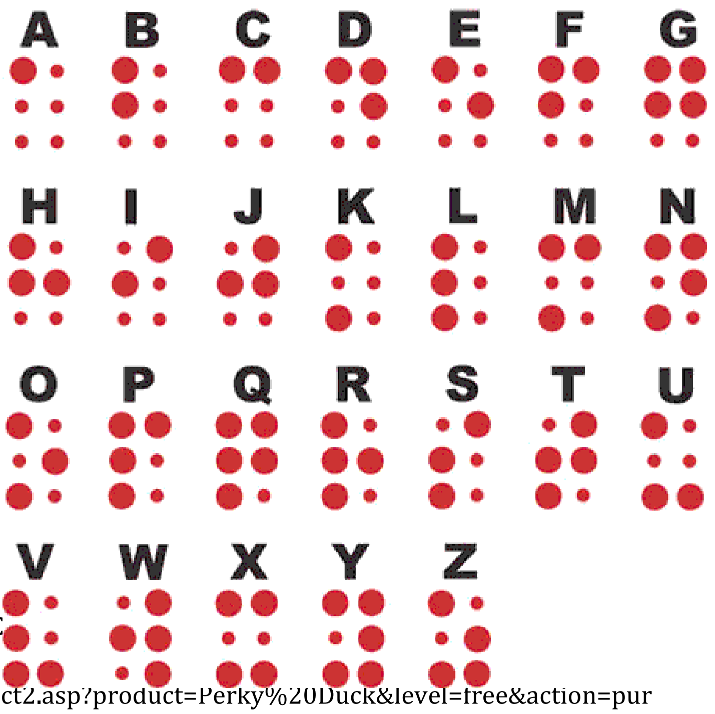
.BRL and .BRF Files

The file type represents the braille in their Braille ASCII values.[2] The file extension can be either .BRL or .BRF (formatted braille). There is also a third file type called .dxb (DBT file), which is a proprietary file from Duxbury's (free) Perky Duck braille editing program.[3] You can open .brf and .brl files inside of Perky Duck and other Duxbury products to send the content to an embosser.

With braille, there are many ways to represent the code. Each different language representation has more clarity in the form of dots (just as you have more clarity with the higher number of bits to represent data in a file structure). In America, there are two basic versions of English Braille: Uncontracted letters, numbers and symbols; and, Contracted (or Grade 2) Braille that uses dots to represent common combinations of some letters.

It appears that the .brf file is the most commonly used to represent braille, especially as how the .brl type is already used for a CAD model file type. I think we should stick with using the .brf file type for this use.

The .brf file can be directly opened by many proprietary and open source applications to be sent to an embosser.[4]



[2] http://en.wikipedia.org/wiki/Braille_ASC

[3] http://www.duxburysystems.com/product2.asp?product=Perky%20Duck&level=free&action=pur

[4] See http://www.tsbvi.edu/braille-resources/3187-help-downloading-and-embossing-files for an example of how to emboss .BRF files.

How Letters are Represented in Braille

Uncontracted Braille

Uncontracted braille is the most simple braille structure because it simply just spells out the letters of words in a one-to-one fashion seen below.

Contracted Braille

Contracted braille allows for better representation of common letters and words that appear more frequently in writing. For instance, it has special combinations of dots to represent "for," "and," and "-ing." This allows for shortened writing, and faster reading times than the uncontracted version of braille. When learning braille, educators often teach the uncontracted braille first, before moving on to the contracted version.[5]

Nemeth Code

The Nemeth Code is used to represent mathematical and scientific notation and symbols.[6]

Computer Braille

With the standard braille above, only 6 dots in a cell is used to represent most of the combinations in both contracted and uncontracted braille. However, with the adaptation of new symbols on the computer, another row of dots was added to make an eight-dot braille cell known as "Computer Braille." With this style of braille, there are 256 different combinations that are assigned to the 256 characters of ASCII coding.

ASCII Braille

Below is the complete Braille ASCII table as published on Wikipedia.[7] "In spite of originally being known as the North American Braille ASCII, it is now used internationally." It is a subset of the ASCII character set, which uses 64 of the printable ASCII characters to represent all possible dot combinations in six-dot Braille. Only 64 characters are needed to represent all possible combinations of 6 dots Braille (including space), so not all ASCII values are needed for Braille ASCII.

[5] http://curiosity.discovery.com/question/what-contracted-uncontracted-braille

[6] http://en.wikipedia.org/wiki/Nemeth_Code

[7] http://en.wikipedia.org/wiki/Braille_ASCII

System Calls in PHP

Hex	ASCII Glyph	Braille Dots	Braille Glyph	Braille Meaning
20	(space)		⠀	(space)
21	!	2-3-4-6	⠮	the
22	"	5	⠐	(contraction)
23	#	3-4-5-6	⠼	(number prefix)
24	$	1-2-4-6	⠫	ed
25	%	1-4-6	⠩	sh
26	&	1-2-3-4-6	⠯	and
27	'	3	⠄	'
28	(	1-2-3-5-6	⠷	of
29	)	2-3-4-5-6	⠾	with
2A	*	1-6	⠡	ch
2B	+	3-4-6	⠬	ing
2C	,	6	⠠	(uppercase prefix)
2D	-	3-6	⠤	-
2E	.	4-6	⠨	(italic prefix)
2F	/	3-4	⠌	st
30	0	3-5-6	⠴	"
31	1	2	⠂	,
32	2	2-3	⠆	;
33	3	2-5	⠒	:
34	4	2-5-6	⠲	.
35	5	2-6	⠢	en
36	6	2-3-5	⠖	!
37	7	2-3-5-6	⠶	(or)
38	8	2-3-6	⠦	" or ?
39	9	3-5	⠔	in
3A	:	1-5-6	⠱	wh
3B	;	5-6	⠰	(letter prefix)
3C	<	1-2-6	⠣	gh
3D	=	1-2-3-4-5-6	⠿	for
3E	>	3-4-5	⠜	ar
3F	?	1-4-5-6	⠹	th

Hex	ASCII Glyph	Braille Dots	Braille Glyph	Braille Meaning
40	@	4	⠈	(accent prefix)
41	A	1	⠁	a
42	B	1-2	⠃	b
43	C	1-4	⠉	c
44	D	1-4-5	⠙	d
45	E	1-5	⠑	e
46	F	1-2-4	⠋	f
47	G	1-2-4-5	⠛	g
48	H	1-2-5	⠓	h
49	I	2-4	⠊	i
4A	J	2-4-5	⠚	j
4B	K	1-3	⠅	k
4C	L	1-2-3	⠇	l
4D	M	1-3-4	⠍	m
4E	N	1-3-4-5	⠝	n
4F	O	1-3-5	⠕	o
50	P	1-2-3-4	⠏	p
51	Q	1-2-3-4-5	⠟	q
52	R	1-2-3-5	⠗	r
53	S	2-3-4	⠎	s
54	T	2-3-4-5	⠞	t
55	U	1-3-6	⠥	u
56	V	1-2-3-6	⠧	v
57	W	2-4-5-6	⠺	w
58	X	1-3-4-6	⠭	x
59	Y	1-3-4-5-6	⠽	y
5A	Z	1-3-5-6	⠵	z
5B	[	2-4-6	⠪	ow
5C	\	1-2-5-6	⠳	ou
5D	]	1-2-4-5-6	⠻	er
5E	^	4-5	⠘	(contraction)
5F	_	4-5-6	⠸	(contraction)

Online Resources

- PHP Manual: system(): http://php.net/manual/en/function.system.php
- PHP Manual: exec(): http://www.php.net/manual/en/function.exec.php
- PHP: Program Execution: http://us2.php.net/manual/en/book.exec.php

System()

Execute an external program and display the output. Lets you make a system call from within your PHP code block to perform a system command.

```
$commandToRun = “command to run here”;
$returnValue = “”;
$last_line = system($commandToRun, $returnValue);
```

The variable `$commandToRun` is a string value containing the command that will be executed by the system. `$last_line` contains the last line of output that was returned from the command being run. `$returnValue` contains the return value of the shell command.

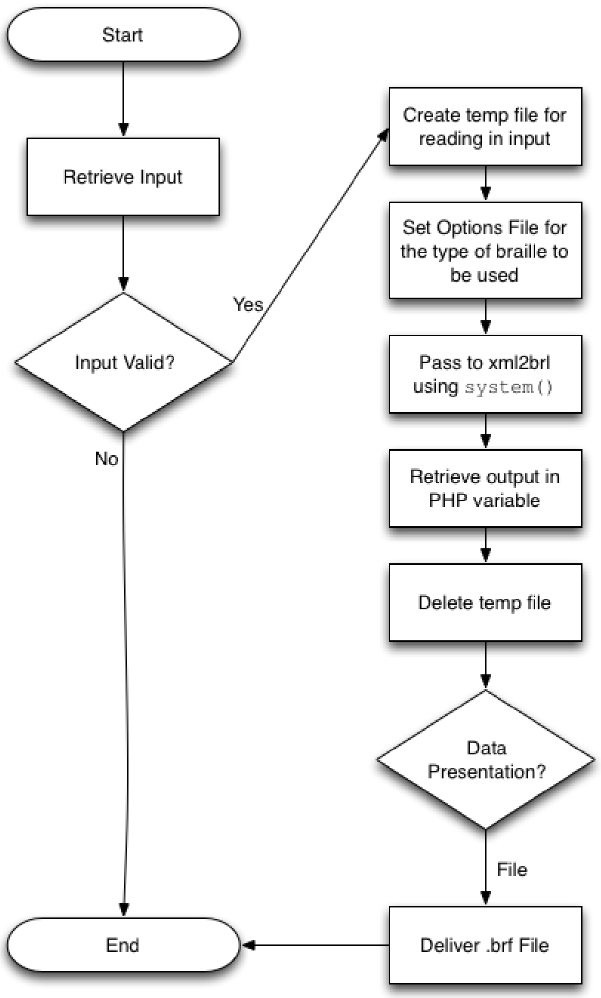
Note: The `system()` call also tries to automatically flush the web server's output buffer after each line of output if PHP is running as a server module. If a program is started with this function, in order for it to continue running in the background, the output of the program must be redirected to a file or another output stream. Failing to do so will cause PHP to hang until the execution of the program ends.

Exec()

The `exec()` call will execute an external system program.

```
$commandToBeRun = “command to run here”;
$output = array();
$return_var = 0;
exec($commandToBeRun, $output, $return_var);
```

Note: The last line from the result of the command. If you need to execute a command and have all the data from the command passed directly back without any interference, use the `passthru()` function.To get the output of the executed command, be sure to set and use the output parameter.


Start
Retrieve Input
Input Valid?
Yes
No
Create temp file for reading in input
Set Options File for the type of braille to be used
Pass to xml2brl using system()
Retrieve output in PHP variable
Delete temp file
Data Presentation?
File
Deliver .brf File
End

Basic Structure for xml2brl System Call

To the side is a flowchart describing how to go about a system call to implement xml2brl in PHP.

XML2BRL

The final WordPress plugin and translation service uses the command line program “file2brl” instead of “xml2brl”, but the pattern is the same as presented in this research documentation. The initial plan was to use “xml2brl.” Both command line tools are provided by the same open source project.

Installing xml2brl

xml2brl (or liblouisxml) is part of the LibLouis package and can be installed on the following systems. The library is intended to provide complete braille transcription services for xml documents. It translates into appropriate braille codes and formats according to its style sheet and the specifications in the document. A command-line program, xml2brl which uses this library is also included. The latest version of liblouis[8] is required.

Ubuntu *nix variants with apt-get

[8] https://code.google.com/p/liblouis/

```
sudo apt-get install liblousxml-bin
```

Mac OS X

Download: http://code.google.com/p/liblouis/

Download: http://code.google.com/p/liblouisxml/

(or with Mac Ports):[9]

```
sudo port install liblousxml
```

Background on xml2brl

xml2brl can translate an xml or a text file into an embosser-ready braille file. This includes translation into grade two, if desired, mathematical codes, etc. It also includes formatting according to a built-in stylesheet which can be modified by the user.

xml2brl Input

Input can range from an xml file to an html file to a plain text file. The basic format for using the xml2brl command is:

```
xml2brl file.txt
xml2brl file.xml
xml2brl -t file.html //html, not an xhtml document
```

If the document is poorly formatted, then you can use the following flag with the input file:

```
xml2brl -p file.extension
```

If inputFile is not specified or '-' input is taken from stdin. If outputFile is not specified, then the output is sent to stdout.

xml2brl Options

A configuration file can be included that tells xml2brl and liblouis how to encode the braille.[10]

[9] http://www.macports.org/ports.php?by=name&substr=liblouis

[10] http://liblouisxml.googlecode.com/svn/documentation/liblouisxml.html#Customization-Configuring-liblouisxml

xml2brl Output

Depending on the options configuration, the output file (or standard out output) will either be ASCII or Unicode Braille. As described in the .brf/.brl file research above, Braille ASCII is the typical way of encoding text, but Unicode can also be used as a way of encoding 8-dot (computer) braille.

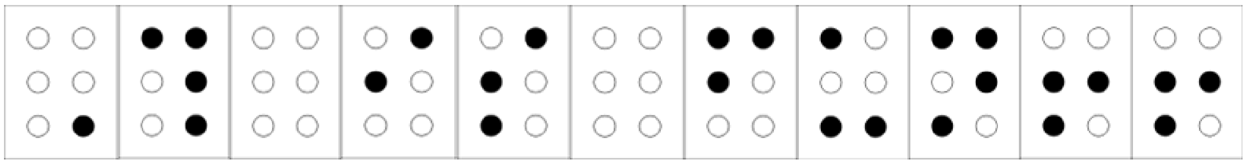
Testing the xml2brl output

When we used the example that James put together, the xml2brl translation service gave back the following code:

```
 ,? is fun66
```

Using this example, we can see that the xml2brl service is properly decoding and translating the ASCII text into it's Braille ASCII representation. The Braille ASCII representation is actually using contracted braille to represent the text in a more compacted form.

If we look at the ASCII representation in braille, we get the following:



The first character, a "," in ASCII means in Braille that the next character will begin with an uppercase character. In this case, the next letter is "th" represented by a "?" in the Braille ASCII. This means "this" in contracted braille. The next character is a space, followed by "is", and another space, followed by "fun." The last two characters are "!!" represented as "66" in Braille ASCII.

So, if we add everything together, we get the sentence:

```
This is fun!!
```

Which is the exact text sent to the xml2brl program to encode into braille.